(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 866 403 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
23.09.1998 Bulletin 1998/39

(51) Int Cl.6: G06F 13/10

(21) Application number: 98301348.3

(22) Date of filing: 24.02.1998

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 17.03.1997 US 820470

(71) Applicant: International Business Machines Corporation
Armonk, N.Y. 10504 (US)

(72) Inventors:
- Mealey, Bruce Gerard
Austin, Texas 78759 (US)
- Swanberg, Randal Craig
Round Rock, Texas 78664 (US)
- Williams, Michael Stephen
Austin, Texas 78759-4530 (US)

(74) Representative: Moss, Robert Douglas
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) Dynamic extension of static device drivers

(57) A method of changing the functionality of a statically bound device driver, by dynamically extending the static device driver using a registered driver extension. The static device driver has a plurality of handlers or functions (such as input/output functions) used to control a device that is connected to or part of the computer system, and the driver extension modifies at least one of these functions, although it can be used to change several, or even all, of the functions. In the embodiment wherein the computer system is a UNIX-type workstation having a kernel residing in the memory, the static device driver is loaded in the kernel and is dynamically extended by providing at least one entry point for the driver extension.

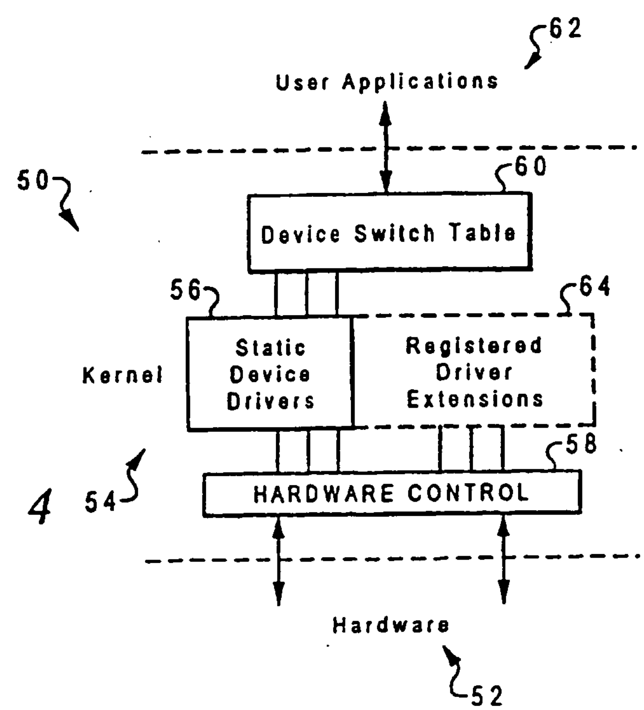

Fig. 4

EP 0 866 403 A1

Description

Technical Field of the Invention

The present invention generally relates to computer systems and more particularly to a method for extending static device driver functionality without requiring rebuilding of the operating system.

Background of the Invention

The basic structure of a conventional computer system **10** is shown in **Figure 1**. The heart of computer system **10** is a central-processing unit (CPU) or processor **12**, which is connected to several peripheral devices, including input/output (I/O) devices **14** (such as a display monitor and keyboard) for the user interface, a permanent memory device **16** (such as a hard disk or floppy diskette) for storing the computer's operating system and user programs, and a temporary memory device **18** (such as random-access memory or RAM) that is used by processor **12** to carry out program instructions. Processor **12** communicates with the peripheral devices by various means, including a bus **20** or a direct channel **22**. Computer system **10** may have many additional components which are not shown, such as serial and parallel ports for connection to, e.g., modems or printers. Those skilled in the art will further appreciate that there are other components that might be used in conjunction with those shown in the block diagram of **Figure 1**; for example, a display adapter connected to processor **12** might be used to control a video display monitor. various types of device drivers (software programs) are used to control the hardware devices.

Computer system **10** also includes firmware **24** whose primary purpose is to seek out and load an operating system from one of the peripherals (usually permanent memory device **16**) whenever the computer is first turned on. The process of seeking out and loading the operating system is referred to as "booting" the computer. Computer system **10** may be designed to allow firmware **24** to re-initialize an operating system without turning the computer off and back on again (a "soft" boot). Firmware **24** is essentially a series of machine instructions which are typically stored in a read-only storage (ROS) device, such as read-only memory (ROM). As shown in the flow chart of **Figure 2**, after power to computer system **10** is turned on, processor **12** begins to execute the firmware instructions and seeks out an operating system (26). If an operating system is found, it is loaded (28) into temporary memory **18**, including any device drivers present in the operating system image, to enable the system to communicate with appropriate hardware. Thereafter, additional device drivers may be dynamically loaded by the operating system (30), for example, if a hardware device is connected to the computer system after the boot sequence. Finally, the operating system allows other application layers to be added, i.e., user software programs (32).

The foregoing description generally applies to any type of operating system, including two popular operating systems known as MSDOS and UNIX (MSDOS is a trademark of Microsoft Corp.; UNIX is a trademark licensed exclusively by X/Open Company Ltd.), but the present invention has particular application to UNIX. UNIX is a multi-user, multi-tasking operating system which is available from a variety of sources with different versions. These include, among others, System V from AT & T, AIX from IBM (AIX is a trademark of International Business Machines Corporation) and Mach from NeXT Computers. **Figure 3** illustrates a typical UNIX workstation **34**. Workstation **34** includes the various hardware components shown in **Figure 1**, and generally represented at **36**, and furthermore includes two software layers, the kernel **38** and the user application layer **40**. Kernel **36** is the lowest level of the operating system and acts as the intermediary between user programs and hardware devices and includes, among other things, device drivers that interface with hardware control **42**. Kernel **38** may include static device drivers **44** which are originally bound with the kernel during initialization and dynamically loaded device drivers **46** which are added to the kernel after initialization. A dynamically loaded device driver **46** can be used for a unique device or can simply be a replacement for a generic static device driver. Both types of device drivers are usually accessed by a buffering mechanism such as a device switch table **48**.

Device drivers are often hardware dependent, which can present difficulties when installing or using particular hardware devices. If an appropriate device driver for a new device is not already present in the kernel, (i.e., statically bound) and if a dynamically loadable driver is not available, then the kernel must be re-bound with a new static device driver. If a dynamically loadable driver is available, then it can easily be loaded as a kernel extension but, in some cases, the system requires certain devices or hardware functions to be provided only via static device drivers bound in the kernel since the facilities they require must exist prior to the ability to load kernel extensions. These functions include, for example, NVRAM, RAMDD, and console device drivers. In these cases, the only way to enhance the functionality or capabilities of the static device driver is to rebuild the base kernel. Similarly, there is no way to modify a static device driver once it has been loaded. For example, there might be a bug (software instruction error) in the static device driver, but it cannot be fixed unless the kernel is rebuilt. It would, therefore, be desirable and advantageous to devise a method of changing the functionality of a statically bound device driver without requiring rebuilding of the kernel or otherwise rebooting the operating system.

Disclosure of the Invention

According to the present invention, there is provid-

ed a method for providing control of a device in a computer system having memory, comprising the steps of: loading a static device driver into the memory of the computer system, the static device driver having means for providing a plurality of functions used to control the device; the method being characterised by the further step of: dynamically extending the static device driver using a driver extension.

According to another aspect of the invention, there is provided a computer system comprising: a hardware device; memory means storing a static device driver used to control said hardware device; processor means for carrying out instructions from said static device driver to control said hardware device; characterised in that the system further comprises: a device driver extension registered with said static device driver such that a function call issued to said hardware device is rerouted by said static device driver to said device driver extension.

Thus, the invention provides an improved method of loading device drivers for a computer system which allows modification of a statically bound device driver without requiring rebuilding of the operating system.

The preferred method provides control of a device in a computer system, generally comprising the steps of loading a static device driver into the memory of the computer system, and thereafter dynamically extending the static device driver using a driver extension which is registered with the static device driver. The static device driver has a plurality of handlers or functions (such as input/output functions) used to control the device, and the driver extension modifies at least one of these functions, although it can be used to change several, or even all, of the functions. In the preferred embodiment wherein the computer system is a UNIX-type workstation having a kernel residing in the memory, the static device driver is within the kernel and is dynamically extended by providing at least one entry point for the driver extension.

Brief Description of the Drawings

The invention will now be described with reference to a preferred embodiment thereof, as illustrated in the accompanying drawings, wherein:

Figure 1 is a block diagram of a prior-art computer-operating system;

Figure 2 is a flow chart illustrating how a computer loads a prior-art operating system with static device drivers bound to the operating system, and then loads dynamically loadable device drivers after loading the operating system;

Figure 3 is a block diagram of a prior art UNIX-type workstation having static and dynamic device drivers;

Figure 4 is a block diagram of a UNIX-type workstation configured according to the present invention with registered driver extensions; and

Figure 5 is a flow chart illustrating how, according to the present invention, a computer loads an operating system with static device drivers that allow registration of driver extensions, and how the static drivers dynamically access the registered extensions.

Detailed Description of the Invention

With reference now to the figures, and in particular with reference to **Figure 4**, there is depicted one embodiment **50** of a UNIX-type workstation according to the present invention. Workstation **50** is generally comprised of the same basic hardware as shown in **Figure 1**, portions of which are indicated at **52**, but workstation **50** has a novel operating system loaded in kernel **54** which allows for registration of static device-driver extensions. Kernel **54** has the static device drivers **56** bound thereto, and includes conventional hardware control **58** that is connected to the outputs of the device drivers. Static device drivers **56** are accessed via a conventional-device switch table **60** that acts as an interface with the user applications **62**. Kernel **54** also includes driver extensions **64** that are "registered" with static device drivers **56**. The driver extensions provide control for hardware control **58**, but are not accessed by the device switch table. Instead, static device drivers **56** register specific handlers or functions (like open, read, ioctl, or other I/O control functions) to provide process entry points for those functions with respect to the particular device being accessed.

This solution allows the functionality of a static device driver to be dynamically extended in order to get the flexibility of a loadable device driver in cases where static boot-time functionality is also required. One such example is that, at a later point during system initialization, a kernel extension could register its own ioctl() handler to be called in response to an ioctl() call to that static device driver. This example could also be expanded to other device-driver entry points. In this manner, a user can change the function of a device driver that is statically bound without rebinding the kernel. This adaptability allows a user to enhance a device driver, e.g., provide hardware-specific differentiators, or to otherwise modify the driver, e.g., fix a bug.

The typical operation of a computer system using registered device extensions is shown in **Figure 5**. After power to the system is turned on (or a soft-boot command is executed), the firmware searches for an operating system to load (**70**). The operating system is then loaded into primary memory, including any static device drivers (**72**). These drivers are configured to recognize extensions based on the device driver entry points (**74**). For example, if a static NVRAM read/write device driver

entry point is called, the driver first checks to see if there are any registered extensions for that entry point and, if so, calls the registered extension. The extension can examine the offset of the NVRAM request and determine if it was in the NVRAM device that it controls. If so, the extension would service the request, but if not, it then would return control to the static driver, which would call the next registered extension or, if appropriate, handle the request itself. After the operating system has been loaded and any driver extensions have been registered, regular user programs are executed (**76**). Then, when any application sends a function call to a static device driver with a registered extension, the static driver reroutes the call to the extension (**78**). In this manner, registration allows the extension to override a particular functionality of, or add new functionality to, the static driver.

In one specific implementation of the present invention, an AIX operating system is modified to provide a machine device driver (/dev/nvram) which is a static device driver providing the base functionality required before it is possible to dynamically load further support. However, via a special extension file registered with the machine device driver, additional functions are provided that are machine-specific. This approach allows common static functions to remain in the base operating system while moving machine-specific functions to appropriate kernel extensions to be dynamically loaded and registered with the machine device driver at run-time.

Unlike dynamically loaded drivers, registered driver extensions **64** are not usually a complete driver, i.e., the extension usually modifies only a portion of the driver functionality. A registered extension could, however, completely replace all functionality for a given static driver, or two or more registered extensions could be used to override all base functionality. Those skilled in the art will appreciate that an operating system constructed in accordance with the present invention can still have dynamically loaded drivers (not shown in **Figure 4**) in addition to registered driver extensions **64**.

Claims

1. A method for providing control of a device (14) in a computer system (10) having memory (18), comprising the step of:

   loading (72) a static device driver (56) into the memory (18) of the computer system, the static device driver having means for providing a plurality of functions used to control the device; the method being characterised by the further step of:

   dynamically extending (74) the static device driver using a driver extension (64).

2. A method as claimed in Claim 1 wherein the static device driver is dynamically extended in response to a function call from a user application.

3. A method as claimed in Claim 1 or Claim 2 wherein the static device driver is loaded as part of the further step (72) of loading an operating system on the computer system.

4. A method as claimed in any preceding claim wherein:

   the computer system is a UNIX-type workstation having a kernel (54) residing in the memory;

   the static device driver is loaded in the kernel; and

   the static device driver is dynamically extended by providing at least one entry point for the driver extension.

5. A method as claimed in any preceding claim wherein the static device driver is dynamically extended by providing (74) a registration in the static device driver for the driver extension.

6. A method as claimed in any preceding claim wherein the driver extension modifies at least one of the plurality of functions of the static device driver.

7. A method as claimed in any preceding claim wherein the driver extension provides a new function for controlling the device.

8. A method as claimed in any preceding claim wherein the driver extension is also loaded in the memory of the computer.

9. A method as claimed in Claim 6 wherein said at least one function is an input/output function.

10. A method as claimed in Claim 6 wherein the static device driver is dynamically extended by registering the driver extension with at least one function.

11. A computer system comprising:

    a hardware device (52);

    memory means (18) storing a static device driver (56) used to control said hardware device;

    processor means (12) for carrying out instructions from said static device driver to control said hardware device; characterised in that the system further comprises:

a device driver extension (64) registered with said static device driver such that a function call issued to said hardware device is rerouted by said static device driver to said device driver extension.

12. A computer system as claimed in Claim 11 wherein:

said static device driver provides a plurality of functions for controlling said hardware device; and

said static device driver is arranged to re-route a function call to said device driver extension for one or more of said functions.

13. A computer system as claimed in Claim 11 or Claim 12 wherein:

the computer system is a UNIX-type workstation having a kernel (54) residing in said memory means;

said static device driver is loaded in said kernel; and

said static device driver is arranged to re-route said function call by providing an entry point for said device driver extension.

14. A computer system as claimed in Claim 11 wherein said device driver extension provides a new function for controlling the device.

15. A computer system as claimed in Claim 13 wherein the kernel includes a hardware control unit (58), and said device driver extension interfaces directly with said hardware control unit.

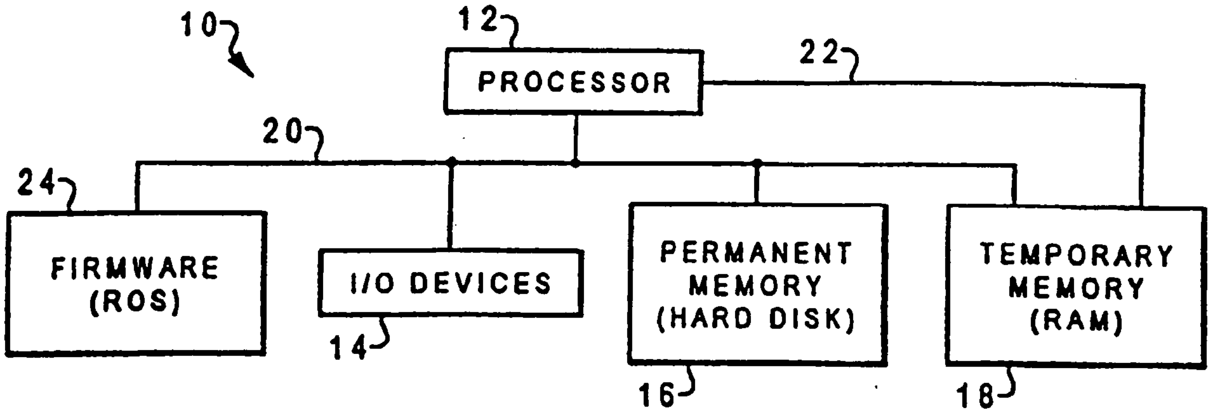

Fig. 1

Prior Art

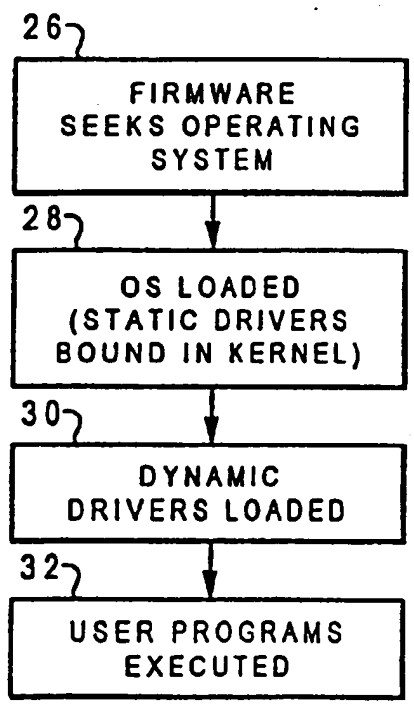

Fig. 2

Prior Art

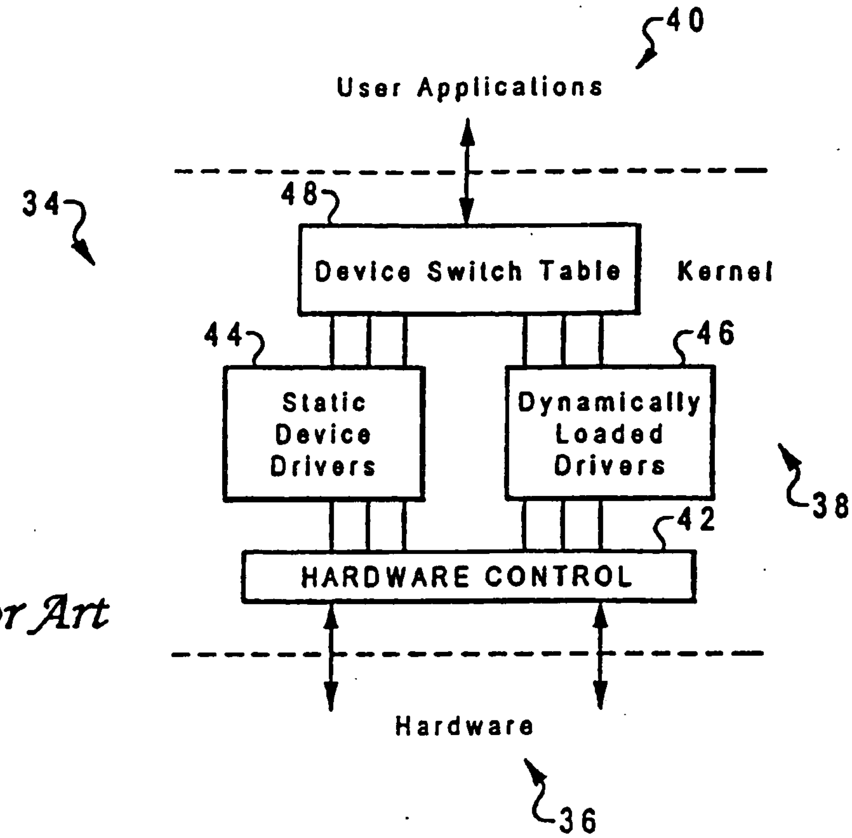

Fig. 3

Prior Art

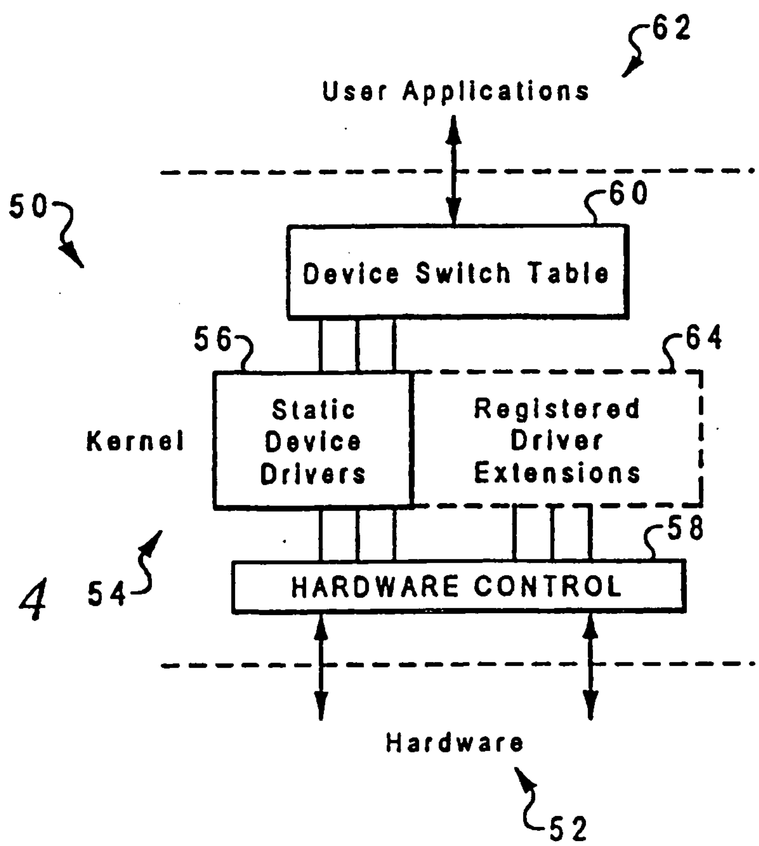

Fig. 4

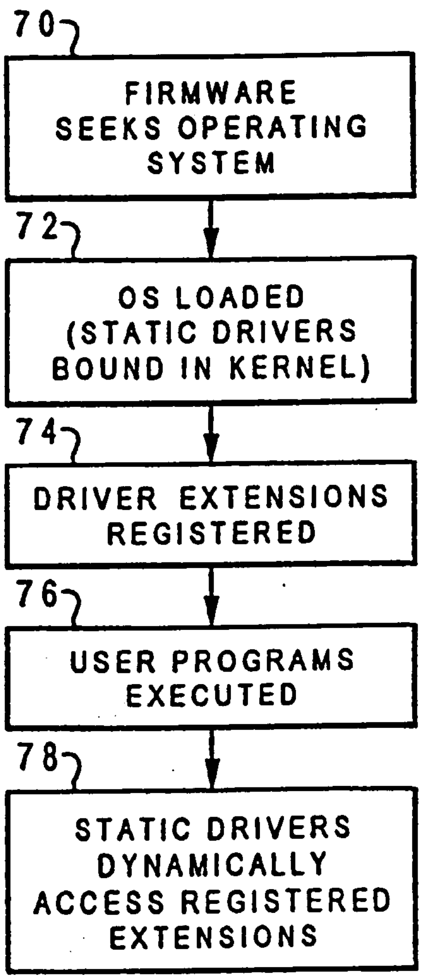
70
FIRMWARE SEEKS OPERATING SYSTEM
72
OS LOADED (STATIC DRIVERS BOUND IN KERNEL)
74
DRIVER EXTENSIONS REGISTERED
76
USER PROGRAMS EXECUTED
78
STATIC DRIVERS DYNAMICALLY ACCESS REGISTERED EXTENSIONS

Fig. 5

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 30 1348

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	"SUPPORT OF A DYNAMICALLY MANAGED DEVICE SWITCH TABLE ON A UNIX OPERATING SYSTEM" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 34, no. 10A, 1 March 1992, pages 130-132, XP000302252	1-15	G06F13/10
A	RUSSINOVICH M ET AL: "EXAMINING VXD SERVICE HOOKING" DR. DOBB'S JOURNAL, vol. 21, no. 5, May 1996, page 32, 34, 36/37 XP002035779	1-15	
A	"DOS Dynamic Device Driver" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 28, no. 12, May 1986, NEW YORK US, page 5249 XP002070656 * the whole document *	1,11	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	7 July 1998	Fonderson, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document